Docket No.: M4065.0503/P503
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
John W. Regnier

Application No.: 10/050,067

Group Art Unit: N/A

Filed: January 17, 2002

Examiner: Not Yet Assigned

For: A FAST ALGORITHM TO EXTRACT FLAT INFORMATION FROM HIERARCHICAL NETLISTS

FIRST PRELIMINARY AMENDMENT

Box Non-Fee Amendment
Commissioner for Patents
Washington, DC 20231

Dear Sir:

Prior to examination on the merits, please amend the above-identified U.S. patent application as follows:

In the Specification

Please replace the figure references to Fig. 5, Fig. 6 and Fig. 6a in the Brief Description of The Drawings on pages 7-8 with the following:

Figure 5a shows a portion of a processing sequence for processing and testing a netlist and assembling an absolute path statement constructed in accordance with an exemplary embodiment the invention;

1426693 v1; %K%D01!.DOC

Figure 5b shows a portion of a processing sequence for processing and testing a netlist and assembling an absolute path statement constructed in accordance with an exemplary embodiment the invention;

Figure 6a shows a portion of an output from one embodiment of the invention;

Figure 6b shows a portion of an output from one embodiment of the invention; and

Figure 7 shows additional detail of the Figure 6a output in accordance with an exemplary embodiment of the invention.

Please amend the specification as follows:

Referring to Fig. 5a and 5b, one exemplary processing sequence that may be used to carry out the invention is disclosed. Processing is initiated by loading a hierarchical netlist 1 into the processing system 71 through either the input device 75 or from storage 73 and the ReadNetlist module 153 reads in hierarchical netlist 1 at processing segment 200. Next, the input hierarchical netlist 1 is parsed by the parseNetlist module 155 storing cell data into variable %CELL 91, instance data into %INST 93 at processing segment 201. In processing segment 203, the getLocalHV module 157 parses the instances (%INST 101) finds local HV nets/markers 103 for each individual cell type (i.e. searches one instance of Fig. 1 ercFlatA 11, ercFlatB 21, ercFlatC 31) and stores local marker and cell data into data structure %HV 103. It should be noted that processing segment 203 could also be performed later in the processing sequence such as, for example, in Fig. 5b after processing segment 219 since segment 221 and 223 use the cell and HV marker. In processing segment 205, the getFlatHV module 159 is called and the top level HV nets/markers 15 are added to the %FLATHV 107 data structure. Next, the getFlatHV module 159 checks to see if there is another instance in the cell which is currently being scanned at processing segment 209. The getFlatHV module 159 will then get the identity or name of the next unscanned child instance in the current cell being scanned at processing segment 211. Note that the next child instance may be a cell instance, a marker or other component such as a resistor. Referring to Fig. 5b, the GetFlatHV module 159 will then determine if the next child instance is a cell at processing segment 213. Referring to Fig. 5a, if the next child instance is not a cell, then processing is branched back to processing segment 207 and then GetFlatHV 159 will again check to see if there is another instance in the current cell being scanned at processing segment 209. If there is another instance, then GetFlatHV 159 will again check to see if the scanned instance is a cell at processing segment 213 (Fig. 5b). This loop from processing segment 213 back to 207 will continue until either there are no other instances (processing segment 209) or the instance being scanned is a cell (processing segment, Fig. 5b, 213).

[0029] Referring to Fig. 5b, if the getFlatHV module 159 determines there is another cell instance within the current cell hierarchy at processing segment 213 (e.g. in the Fig. 1 schematic, there are multiple child cells within ercFlatC 31), then the getFlatHV module 159 will then determine the net mapping for the child cell from the child's parent to the child by calling the netNetMap module 161 at processing segment 215 which will store net mapping information. The getFlatHV module 159 will then determine if the child cell has been scanned previously at processing segment 217 by determining if the selected cell's instance name is stored in the SEEN data structure 111. If the child cell has not been scanned, then getFlatHV module 159 will mark the cell as visited in the SEEN 111 data structure at processing segment 219. In processing segment 221, the local child cell flat data net and marker entries will be added to the %FlatHV data structure 107. Next, the local child cell flat data net, marker and component entries will be added to the %FlatHVcache 109 data structure at processing segment 223. The %FlatHVcache 109 data structure holds instance data structures which are retrieved at subsequent processing segments for addition to upper level hierarchy flat data stored during a netlist traversal sequence leading up to the currently selected cell instance. At processing segment 225, the getFlatHV module 159 calls itself recursively thereby returning to processing segment 207 with the currently designated child cell as a top cell or traversal starting point.

[0031] When the getFlatHV module 159 determines at processing segment 209 that there are no additional instances along the hierarchical traversal path being processed, then getFlatHV 159 will either terminate completely at processing segment 235 or the current getFlatHV 159 module will terminate and return to the getFlatHV module 159 that previously called it at processing segment 227. Again, child cell data will be copied back to parent cell data elements within the %FlatHVcache data structure 109 to facilitate further processing and tracking of the current point of the netlist traversal. Referring to Fig. 5a, in such a manner, the progressively called getFlatHV 159 modules which executed a copy of itself at processing segment 225 will begin to complete processing and reverse the netlist traversal to the cell instance which is above the current cell which is being mapped. The processing sequence of 207 to 213 will be re-executed until another child cell instance is encountered. The processing sequence from 214 to 233 will be re-executed when a traversal selects a cell instance which matches an instance identifier in the SEEN 111 data structure, at which point the getFlatHV module 159 will retrieve the matching flat data and append the flat data to the higher level flat data corresponding to the currently netlist traversal sequence path. The higher level flat data is thus hierarchically and traversally interconnected with the appended data through the traversal sequence. In this manner, instance by instance parsing, mapping and storing is avoided saving considerable time. Referring to Fig. 5a and 5b, complete mapping will still have to be accomplished for cells which have not yet been encountered during traversal of the netlist in processing segments 217 through 225. Processing is complete when all recursively called modules have terminated and there are no further instances at processing segment 235.

[0032] Referring to Fig. 6a, output for an exemplary embodiment of the invention is shown. The first set of entries 301 under the %HV header correspond to step 203 data. The second set of entries 303 under the %FLATHV header correspond to the final flat data which is being created in the %FlatHV data structure 107 as flat data segments (and their corresponding highest level absolute net paths) are assembled during traversal. Referring to Fig. 6b, the third set of entries 305 under the %FLATHVCACHE header correspond to the data being accumulated in the %FlatHVcache data structure 109.

[0033] Referring to Fig. 7 and also to the hierarchical representation in Fig. 1, flat data output under the %FLATHV header 303 is explained. The first set of entries 401 indicate the top path keys on the left which defines the upper hierarchical point of the flat data path sequence. The entries on the right of the dashes 403 indicate the flat data following the top net identifier on the left to a particular flat path search target point at the end of the traversal path, a HV marker in this case 417. Referring to Fig. 7, the first element "/" 405 indicates the top cell which is being scanned, in this case ercFlatTop IO 2. Top_2 407 refers to the top net starting point which is being mapped. The "/X0" entry 411 refers to the first search marker, a HV marker in this case, which was encountered while traversing along the top_2 net (Fig. 1, 5). Next, the "/I0" entry 413 refers to the ercFlatC I0 cell in figure 1 (Fig. 1, 31). Next, the "/I0" entry 415 refers to instance zero of ercFlatA (fig. 1, 35) along the traversed net. Next, the "/X0" entry 417 refers to the final search target along the traversed net, in this case a HV marker within ercFlatA (Fig. 1, 35). In this manner, flat data is described from any number of top nets to a search target element.

REMARKS/ARGUMENTS

The changes to the specification are being made to conform the specification to the enlarged formal drawings which have been submitted. The formal drawings differ from the informal drawings in that Fig. 5 is now Fig. 5a and Fig. 5b. Fig. 6 is now Fig. 6a and Fig. 6b. The previous Fig. 6a is now Fig. 7. Attached hereto is a marked-up version of the changes made to the specification and claims by the current amendment. The attached page is captioned "**Version with markings to show changes made.**"

In view of the above, each of the presently pending claims in this application is believed to be in immediate condition for allowance. Accordingly, the Examiner is respectfully requested to pass this application to issue.

Dated: April 8, 2002

Respectfully submitted,

By [illegible]
Thomas J. D'Amico
Registration No.: 28,371
DICKSTEIN SHAPIRO MORIN & OSHINSKY LLP
2101 L Street NW
Washington, DC 20037-1526
(202) 785-9700
Attorneys for Applicant

Version With Markings to Show Changes Made

Please replace the figure references to Fig. 5, Fig. 6 and Fig. 6a in the Brief Description of The Drawings on pages 7-8 with the following:

[Figure 5] Figure 5a shows a portion of a processing sequence for processing and testing a netlist and assembling an absolute path statement constructed in accordance with an exemplary embodiment the invention;

Figure 5b shows a portion of a processing sequence for processing and testing a netlist and assembling an absolute path statement constructed in accordance with an exemplary embodiment the invention;

[Figure 6] Figure 6a shows a portion of an output from one embodiment of the invention; [and]

Figure 6b shows a portion of an output from one embodiment of the invention; and

[Figure 6a] Figure 7 shows additional detail of the [figure 6] Figure 6a output in accordance with an exemplary embodiment of the invention.

Please amend the specification as follows:

[0028] Referring to [Fig. 5] Fig. 5a and 5b, one exemplary processing sequence that may be used to carry out the invention is disclosed. Processing is initiated by loading a hierarchical netlist 1 into the processing system 71 through either the input device 75 or from storage 73 and the ReadNetlist module 153 reads in hierarchical netlist 1 at processing segment 200. Next, the input hierarchical netlist 1 is parsed by the parseNetlist module 155 storing cell data into variable %CELL 91, instance data into %INST 93 at processing segment 201. In processing segment 203, the getLocalHV module 157 parses the instances (%INST 101) finds local HV nets/markers 103 for each individual cell type (i.e. searches one instance of Fig. 1 ercFlatA 11, ercFlatB 21, ercFlatC 31) and stores local marker and cell data into data structure %HV 103. It should be noted that processing segment 203 could also be performed later in the processing sequence such as, for example, in Fig. 5b after processing segment 219 since segment 221 and 223 use the cell and HV marker. In processing segment 205, the getFlatHV module 159 is called and the top level HV nets/markers 15 are added to the %FLATHV 107 data structure. Next, the getFlatHV module 159 checks to see if there is another instance in the cell which is currently being scanned at processing segment 209. The getFlatHV module 159 will then get the identity or name of the next unscanned child instance in the current cell being scanned at processing segment 211. Note that the next child instance may be a cell instance, a marker or other component such as a resistor. Referring to Fig. 5b, the [The] GetFlatHV module 159 will then determine if the next child instance is a cell at processing segment [211] 213. Referring to Fig. 5a, if [If] the next child instance is not a cell, then processing is branched back to processing segment 207 and then GetFlatHV 159 will again check to see if there is another instance in the current cell being scanned at processing segment 209. If there is another instance, then GetFlatHV 159 will again check to see if the scanned instance is a cell at processing segment 213 (Fig. 5b). This loop from processing segment 213 back to 207 will continue until either there are no other instances (processing segment 209) or the instance being scanned is a cell (processing segment, Fig. 5b, 213).

[0029] Referring to Fig. 5b, if [If] the getFlatHV module 159 determines there is another cell instance within the current cell hierarchy at processing segment 213 (e.g. in the Fig. 1 schematic, there are multiple child cells within ercFlatC 31), then the getFlatHV module 159 will then determine the net mapping for the child cell from the child's parent to the child by calling the netNetMap module 161 at processing segment 215 which will store net mapping information. The getFlatHV module 159 will then determine if the child cell has been scanned previously at processing segment 217 by determining if the selected cell's instance name is stored in the SEEN data structure 111. If the child cell has not been scanned, then getFlatHV module 159 will mark the cell as visited in the SEEN 111 data structure at processing segment 219. In processing segment 221, the local child cell flat data net and marker entries will be added to the %FlatHV data structure 107. Next, the local child cell flat data net, marker and component entries will be added to the %FlatHVcache 109 data structure at processing segment 223. The %FlatHVcache 109 data structure holds instance data structures which are retrieved at subsequent processing segments for addition to upper level hierarchy flat data stored during a netlist traversal sequence leading up to the currently selected cell instance. At processing segment 225, the getFlatHV module 159 calls itself recursively thereby returning to processing segment 207 with the currently designated child cell as a top cell or traversal starting point.

[0031] When the getFlatHV module 159 determines at processing segment 209 that there are no additional instances along the hierarchical traversal path being processed, then getFlatHV 159 will either terminate completely at processing segment 235 or the current getFlatHV 159 module will terminate and return to the getFlatHV module 159 that previously called it at processing segment 227. Again, child cell data will be copied back to parent cell data elements within the %FlatHVcache data structure 109 to facilitate further processing and tracking of the current point of the netlist traversal. Referring to Fig. 5a, in [In] such a manner, the progressively called getFlatHV 159 modules which executed a copy of itself at processing segment 225 will begin to complete processing and reverse the netlist traversal to the cell instance which is above the current cell which is being mapped. The processing sequence of 207 to 213 will be re-executed until another child cell instance is encountered. The processing sequence from 214 to 233 will be re-executed when a traversal selects a cell instance which matches an instance identifier in the SEEN 111 data structure, at which point the getFlatHV module 159 will retrieve the matching flat data and append the flat data to the higher level flat data corresponding to the currently netlist traversal sequence path. The higher level flat data is thus hierarchically and traversally interconnected with the appended data through the traversal sequence. In this manner, instance by instance parsing, mapping and storing is avoided saving considerable time. Referring to Fig. 5a and 5b, complete [Complete] mapping will still have to be accomplished for cells which have not yet been encountered during traversal of the netlist in processing segments 217 through 225. Processing is complete when all recursively called modules have terminated and there are no further instances at processing segment 235.

[0032] Referring to [Figure 6] Fig. 6a, output for an exemplary embodiment of the invention is shown. The first set of entries 301 under the %HV header correspond to step 203 data. The second set of entries 303 under the %FLATHV header correspond to the final flat data which is being created in the %FlatHV data structure 107 as flat data segments (and their corresponding highest level absolute net paths) are assembled during traversal. Referring to Fig. 6b, the [The] third set of entries 305 under the %FLATHVCACHE header correspond to the data being accumulated in the %FlatHVcache data structure 109.

[0033] Referring to [Fig. 6a] Fig. 7 and also to the hierarchical representation in Fig. 1, flat data output under the %FLATHV header 303 is explained. The first set of entries 401 indicate the top path keys on the left which defines the upper hierarchical point of the flat data path sequence. The entries on the right of the dashes 403 indicate the flat data following the top net identifier on the left to a particular flat path search target point at the end of the traversal path, a HV marker in this case 417. Referring to [Fig. 6a] Fig. 7, the first element "/" 405 indicates the top cell which is being scanned, in this case ercFlatTop IO 2. Top_2 407 refers to the top net starting point which is being mapped. The "/X0" entry 411 refers to the first search marker, a HV marker in this case, which was encountered while traversing along the top_2 net (Fig. 1, 5). Next, the "/I0" entry 413 refers to the ercFlatC I0 cell in figure 1 (Fig. 1, 31). Next, the "/I0" entry 415 refers to instance zero of ercFlatA (fig. 1, 35) along the traversed net. Next, the "/X0" entry 417 refers to the final search target along the traversed net, in this case a HV marker within ercFlatA (Fig. 1, 35). In this manner, flat data is described from any number of top nets to a search target element.